

Making a Faster Cryptanalytic Time-Memory Trade-Off

Создание ускоренного криптоаналитического метода «Time-Memory Trade-Off» (компромисс время-память)

Philippe Oechslin

Laboratoire de Securite et de Cryptographie (LASEC)

Ecole Polytechnique Federale de Lausanne

Faculte I&C, 1015 Lausanne, Switzerland

philippe.oechslin@epfl.ch

Перевод на русский язык: Александр Полуэктов, ©2006, InsidePro,

<http://www.OutsidePro.com/rus>

Версия документа: 1.0

Краткий обзор

В 1980 году Martin Hellman описал компромиссное решение криптоаналитической задачи, сокращающее время криптоанализа за счет использования предварительно рассчитанных таблиц, хранящихся в памяти. Эту технологию в 1982 г. улучшил Rivest путем введения ключевых точек, которые значительно сократили количество обращений к памяти в ходе криптоанализа. Эта улучшенная технология с тех пор исследовалась достаточно широко, но новых оптимизаций никогда не вносилось. Мы предлагаем новый метод предварительного расчета данных, вдвое сокращающий количество вычислений в ходе криптоанализа. Более того, поскольку этот метод не использует ключевые точки, то сокращается объем служебных данных, что существенно сокращает количество вычислений. В качестве примера приведем перебор LM-хэшей к паролям Microsoft Windows. Располагая объемом данных в 1.4Gb (два CD-диска), мы можем взломать 99.9% всех хэшей к буквенно-цифровым паролям за 13.6 секунд, в то время как по действующей методике с использованием ключевых точек это займет 101 секунду. Мы докажем, что преимущество может быть еще ощутимее в зависимости от заданных параметров.

1 Введение

Криптоаналитический исчерпывающий перебор требует значительных затрат вычислительной мощности или времени. Когда один и тот же перебор требуется выполнять неоднократно, можно заранее выполнить исчерпывающий перебор и сохранить результаты в памяти. Однократно выполненный предварительный расчет позволяет осуществлять перебор практически мгновенно. Увы, этот метод практически неприменим из-за огромных затрат памяти. В [4] Hellman предлагает компромисс между затратами памяти и временем на перебор. В случае с криптосистемой с N ключей этот метод может восстановить ключ посредством $N^{2/3}$ операций (с использованием $N^{2/3}$ единиц памяти). Типичное применение этого метода – восстановление ключа по известному исходному и зашифрованному тексту. Одна из сфер его применения – какая-либо слабая система шифрования, где взломщик может угадать первые несколько байт данных (например, "#include <stdio.h>"). Другая сфера – хэши паролей. Многие операционные системы генерируют хэши от паролей пользователей путем шифрования простого текста с использованием пароля пользователя в качестве ключа, сохраняя результат под видом хэша к паролю. Опять же, если система создания хэшей к паролям продумана плохо, то исходный текст и метод

шифрования всегда будут одни и те же для всех паролей. *Прим. переводчика: использование SALT'овых алгоритмов, т.е. алгоритмов, где для генерации хэша к паролям различных пользователей используется еще и различная «привязка» – т.н. SALT, т.е. последовательность из случайных символов, существенно усложняет восстановление паролей и делает атаку по предварительно рассчитанным данным абсолютно бесполезной. В качестве примера можно также привести и HMAC-алгоритмы хэширования, где при хэшировании используется уникальный HMAC-ключ. В таком случае хэши к паролям могут быть заранее рассчитаны и использованы в методе "time-memory trade-off" (компромисс время-память). Этот метод (с нашими улучшениями или же без них) основан на теории вероятностей. Успех при этом не гарантируется и напрямую зависит от соотношения распределения времени и памяти для криптоанализа.*

1.1 Оригинальный метод

Дан фиксированный текст P_0 и соответствующий ему зашифрованный текст C_0 . Метод пытается найти ключ $k \in N$, который использовался для шифрования простого текста с помощью шифра S . Таким образом, мы имеем:

$$C_0 = S_k(P_0)$$

Попробуем предварительно сгенерировать все возможные варианты зашифрованного текста, шифруя его всеми возможными для N ключами. Зашифрованные тексты организуются в цепочки, первый и последний элемент которых хранится в памяти. Хранение только первого и последнего элементов является операцией, ведущей к компромиссу (сохраняя память за счет временных затрат на криптоанализ). Цепочки создаются посредством операции сокращения R , получающей ключ из зашифрованного текста. Сокращение возникает из-за того, что зашифрованный текст длиннее ключа. Последовательное применение шифра S и функции сокращения R приводит к созданию цепочек переменных ключей и зашифрованных текстов:

$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_{i+1}$$

Последовательность $R(S_k(P_0))$ пишется $f(k)$ и генерирует ключ из ключа, что ведет к созданию цепочек ключей:

$$k_i \xrightarrow{f} k_{i+1} \xrightarrow{f} k_{i+2} \rightarrow \dots$$

Создается m цепочек длиной t и их начальные и конечные элементы сохраняются в таблице. Имея зашифрованный текст C , мы можем попытаться установить – нет ли в числе ключей, использованных при генерировании таблицы, ключа генерации C . Для этого нам потребуется создать цепочку ключей, начиная с $R(C)$ до длины t . Если C окажется полученным с помощью ключа, использованного при создании таблицы, то мы в итоге получим ключ, соответствующий последнему ключу соответствующей цепочки. Этот последний ключ был сохранен в памяти вместе с первым ключом цепочки. Используя первый ключ цепочки, возможно восстановить всю цепочку – в частности, ключ, стоящий перед $R(C)$. Это и есть тот ключ, который использовался для генерации C , т.е. искомым.

К сожалению, существует вероятность того, что цепочки, начинающиеся с разных ключей, пересекаются и сливаются. Это происходит вследствие того, что функция R является произвольным сокращением пространства зашифрованного текста до пространства ключей.

Чем больше таблица, тем выше возможность слияния новой цепочки с уже существующей. Каждое слияние сокращает количество конкретных ключей, в действительности охваченных таблицей. Вероятность нахождения ключа с использованием таблицы на m рядов t ключей приведена в оригинале [4] и

составляет:

$$P_{table} \geq \frac{1}{N} \sum_{i=1}^m \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1} \quad (1)$$

Эффективность единичной таблицы резко снижается с увеличением ее размера. Для повышения шанса на успех лучше генерировать множественные таблицы с различными функциями сокращения в каждой.

Вероятность успеха с использованием l таблиц в таком случае определяется:

$$P_{success} \geq 1 - \left(1 - \frac{1}{N} \sum_{i=1}^m \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}\right)^l \quad (2)$$

Цепочки различных таблиц могут пересекаться, но не сливаются, поскольку в различных таблицах применялись различные функции сокращения.

1.1.1 Ложные тревоги

При поиске ключа по таблице нахождение подходящей конечной точки не означает существования ключа в таблице. В действительности, ключ может являться частью цепочки с идентичным конечным звеном, не включенной в таблицу. В таком случае генерация цепочки с сохраненного начального звена не приведет к получению ключа, к которому относится, т.е. возникнет ложная тревога. Ложные тревоги случаются и тогда, когда ключ принадлежит к цепочки, которая, хотя и включена в таблицу, но сливается с другими цепочками таблицы. В таком случае одному и тому же конечному звену соответствует несколько начальных звеньев и для окончательного установления ключа придется сгенерировать несколько цепочек.

1.2 Существующая работа

В [2] Rivest предлагает использовать выделенные точки (далее по тексту – ВТ) как конечные звенья цепочек. Выделенные точки – это точки, для которых истинно какое-либо простое условие (например, первые десять бит ключа – нулевые). Все конечные точки, сохраняемые в памяти, являются выделенными точками. Получив первый зашифрованный текст, мы можем генерировать цепочку ключей до нахождения выделенной точки и лишь затем искать ее в памяти. Это значительно сокращает количество обращений к памяти. Все последующие публикации используют именно этот способ оптимизации.

[6] описывает пути оптимизации параметров таблицы t , m и l для максимального сокращения общих затрат метода, основанного на затратах памяти и механизмов обработки данных.

[5] демонстрирует, что параметры таблицы могут быть настроены таким образом, чтобы повышать вероятность успеха, не повышая при этом потребность в памяти или во времени, затрачиваемого на криптоанализ. В действительности это компромисс между временем, затрачиваемым на предварительный расчет, и вероятностью успеха. Однако, вероятность успеха не может увеличиваться произвольно.

Vorst отмечает в [1], что выделенные точки также обладают двумя преимуществами:

- Позволяют запустить цикл поиска. Если выделенная точка не найдена после перечисления заданного количества ключей, то цепочка считается содержащей цикл и может быть отброшена. В результате ни одна цепочка не содержит циклов.

- Слияния легко обнаруживаются, поскольку две сливающиеся цепочки будут иметь одну и ту же конечную точку (следующая выделенная точка после слияния). Поскольку конечные точки в любом случае сортируются, слияния выявляются без дополнительных затрат. [1] предполагает, что таким образом легко создаются бесконфликтные таблицы коллизий без значительных расходов. Сливающиеся цепочки попросту отбрасываются, а для их замены генерируются новые. Создание таблиц без сливающихся цепочек – еще один компромисс, а именно сокращение затрат памяти за счет дополнительных предварительных расчетов.

Наконец, [7] отмечает, что все предыдущие расчеты, приведенные в предшествовавших работах, основаны на оригинальном методе Hellman'a и использование метода выделенных точек может дать другие результаты благодаря изменению длины цепочки. Они представляют детальный анализ, дублированный эмуляцией на сконструированном для специальных целей БМК.

Вариант компромисса Hellman'a представлен Fiat и Noar в [3]. Хотя этот компромисс менее эффективен, он может быть четко анализирован и, вероятно, способен инвертировать любую функцию.

2 Результаты оригинального метода

2.1 Ограничения и параметры

Компромисс времени-памяти требует указания трех параметров: длина цепочек t , количество цепочек в таблице m и количество созданных таблиц l .

Эти параметры устанавливаются в соответствии с ограничениями памяти M , временем на криптоанализ T и вероятностью успеха $P_{success}$. Ограничение на вероятность успеха дано уравнением 2. Ограничение по памяти M вытекает из количества цепочек в таблице m , количества таблиц l и количеством памяти m_0 , необходимой для сохранения начальных и конечных точек (8 байт в наших экспериментах). Ограничение по времени T вытекает из средней длины цепочек t , количества таблиц l и скорости шифрования текста $1/t_0$ (в нашем случае 700000/сек). Данные ограничения – это худший случай, когда приходится пройти все таблицы, но при этом не учитывается время, потраченное на ложные тревоги.

$$M = m \times l \times m_0 \qquad T = t \times l \times t_0$$

Рисунок 1 иллюстрирует ограничения проблемы взлома буквенно-цифровых паролей Microsoft Windows (сложность 2^{37}). Плоскость верхнего левого графика – ограничение памяти. Решения, удовлетворяющие этим пределам, находятся под поверхностью. Плоскость нижнего левого графика – временные границы. Решение также должно находиться под этой поверхностью, чтобы соответствовать нашим ограничениям. Правый график показывает границы для шанса успеха в 99.9%, комбинируя два предыдущих ограничения.

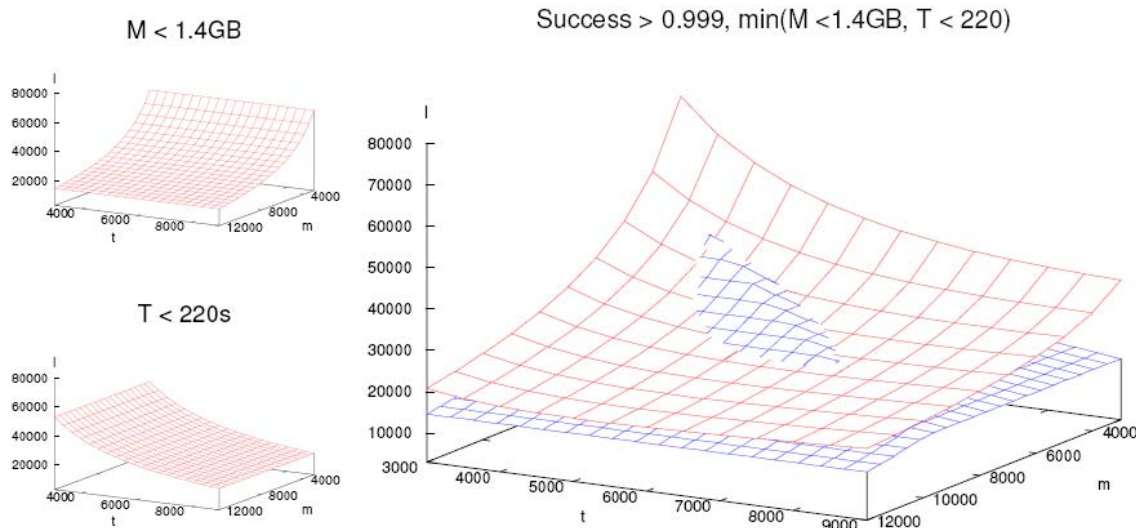


Рисунок 1. Пространство решений с шансом успеха 99.9%, объемом памяти 1.4 GB и пределом в 220 секунд в нашем примере.

Для соответствия всем трем ограничениям параметры решения должны лежать под выступающей поверхностью в центре графика (ограничения по времени и по количеству памяти) и над остальной поверхностью (вероятность успеха). Этот рисунок наглядно иллюстрирует содержание документа [5] – то, что вероятность успеха может быть увеличена без увеличения затрат памяти или времени. Все точки на выступе в центре графика одновременно отвечают требованиям по времени на криптоанализ и по количеству требуемой памяти, однако некоторые из них в большей степени удалены от границы вероятности успеха, нежели другие. Таким образом, вероятность успеха может быть оптимизирована при сохранении того же количества данных и времени на криптоанализ, что является результатом [5]. Мы можем шагнуть даже дальше, чем авторы документа [5], и утверждаем, что оптимальная точка должна лежать на выступе совпадения ограничений времени и памяти, удовлетворяющего условию $t/m = T/M$. Это сокращает поиск оптимального решения на одно измерение.

3 Новая структура таблицы для улучшения результатов

Основное ограничение оригинального метода состоит в том, что, когда две цепочки пересекаются в одной таблице, то они сливаются. Мы предлагаем новый тип цепочек, которые могут пересекаться в одной таблице, при этом не сливаясь.

Мы называем наши цепочки радужными (rainbow). Для каждого звена цепочки используется функция последовательной свертки. Они начинаются со свертки 1 и заканчиваются функцией свертки $t - 1$. Таким образом, при пересечении двух цепочек они сливаются только в том случае, когда коллизия (пересечение) оказывается в одной и той же позиции в обеих цепочках. Если коллизия оказывается не на той же позиции, то обе цепочки продолжают с различными функциями свертки и, следовательно, не сливаются. Для цепочек длиной t шанс слияния при коллизии равен всего лишь $1/t$. Вероятность успеха в пределах одной таблицы размера $m \times t$ вычисляется так:

$$P_{table} = 1 - \prod_{i=1}^t \left(1 - \frac{m_i}{N}\right) \quad (3)$$

где

$$m_1 = m$$

и

$$m_{n+1} = N \left(1 - e^{-\frac{m_n}{N}} \right)$$

Вывод формулы вероятности успеха приводится в приложении. Любопытно, что вероятность успеха в Rainbow-таблицах могут быть прямо соотнесены с этой вероятностью у классических таблиц. Действительно, вероятность успеха для l классических таблиц размера $m \times t$ примерно равна этому показателю для одной Rainbow-таблицы размером $mt \times t$. В обоих случаях таблицы охватывают mt^2 ключей с t различных функций свертки. В каждой точке в пределах набора mt ключей (одна классическая таблица или столбец в Rainbow-таблице) коллизия приводит к слиянию, в то время как коллизия с остальными ключами к слиянию не приводит. Соотношение t таблиц размера $m \times t$ и Rainbow-таблицы показано на рисунке 2. Сопоставление вероятности успеха – на рисунке 3. Обратите внимание, что оси были размечены для создания такой же шкалы, что и в классическом случае с рисунком 1. Может показаться, что Rainbow-таблицы имеют несколько больший показатель этой вероятности, однако это результат того, что в первом случае расчетная вероятность успеха – точная диапазон успеха, тогда как в последнем случае – это ее нижний предел.

Чтобы найти ключ в Rainbow-таблице, мы действуем следующим образом: Сначала мы применяем R_{n-1} к зашифрованному тексту и ищем результат в конечных точках таблицы. Если конечная точка найдена, мы знаем, как достроить цепочку, используя соответствующую начальную точку. Если конечная точка не найдена, мы проверяем – не найдем ли мы его, применив R_{n-2} , f_{n-1} на случай, если ключ находится во втором с конца столбце таблицы. Затем мы пробуем применить R_{n-3} , f_{n-2} , f_{n-1} и т.д. Таким образом, общее число необходимых нам вычислений составляет $t(t-1)/2$. Это вдвое меньше, чем при использовании классического метода. В действительности, нам требуется t^2 вычислений, чтобы найти соответствующие t таблиц размером $m \times t$.

Rainbow-цепочки обладают некоторыми преимуществами по отношению к цепочкам, заканчивающимся выделенными точками и при этом они лишены их ограничений:

- В сравнении с оригинальным методом Hellman'a, количество обращений к таблице делится на коэффициент t .
- Слияния Rainbow-цепочек приводит к возникновению идентичных конечных точек и потому они могут быть выявлены, как и в случае с выделенными точками. Таким образом, Rainbow-цепочки могут использоваться и для создания таблиц без слияний. Обратите внимание, однако – это не означает, что в таблицах нет коллизий!
- Rainbow-цепочки не образуют петель (циклов), поскольку каждая функция свертки вызывается только один раз. Этот способ лучше, чем обнаружение и устранение циклов, как было описано выше, поскольку мы не тратим время на прохождение и отбрасывание циклов, а диапазон наших цепочек не сокращается за счет неохватываемых циклов.
- Rainbow-цепочки обладают постоянной длиной, тогда как цепочки, заканчивающиеся выделенными точками, различаются по длине. Как мы узнаем из раздела 4.1, это сокращает количество ложных тревог, а также лишние затраты, связанных с этим. Этот эффект может оказаться еще более значимым (нежели только в 2 раза) за счет структуры таблицы.

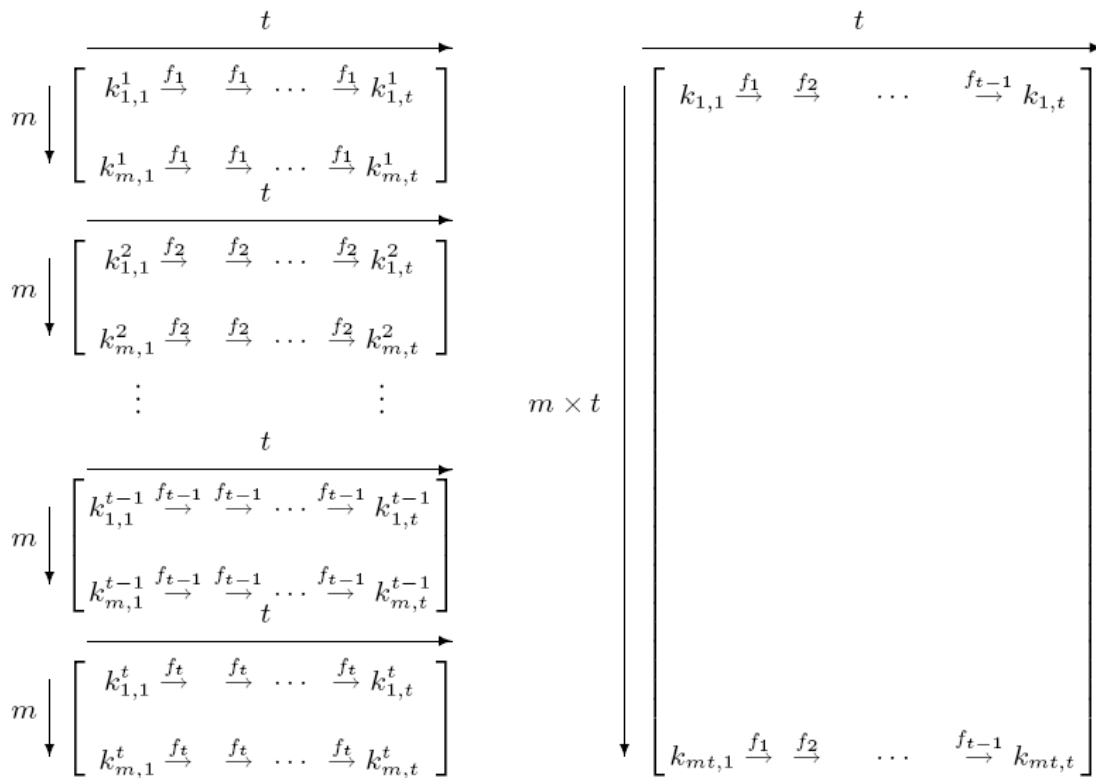


Рисунок 2. t классических таблиц размера $m \times t$ слева и одна Rainbow-таблица размера $mt \times t$ – справа. В обоих случаях слияния могут произойти в пределах группы mt ключей, а коллизии – в оставшихся $m(t - 1)$ ключах. Поиск ключа в Rainbow-таблице требует вдвое меньше операций, чем в классических таблицах.

4 Результаты экспериментов

В качестве примера мы взяли взлом паролей Microsoft Windows, поскольку это имеет вполне реальное практическое применение и может быть реализовано на любом обычном компьютере. Хэш к паролю, который мы пробовали взломать – это LM-хэш, все еще поддерживаемый всеми версиями Microsoft Windows в целях совместимости с предыдущими версиями. Этот хэш формируется путем разделения 14-символьного пароля на две части по семь символов. В каждой части символы нижнего регистра переводятся в верхний регистр и затем каждая часть (по отдельности) используется в качестве ключа для шифрования простого текста с применением стандарта шифрования DES. Это приводит к образованию двух хэшей размером по 8 байт, которые сливаются, формируя один 16-байтовый LM-хэш. Таким образом, обе части LM-хэша могут подвергаться криптоанализу по отдельности, что существенно упрощает их взлом.

Success > 0.999 and min(Memory <1.4GB, Time < 110)

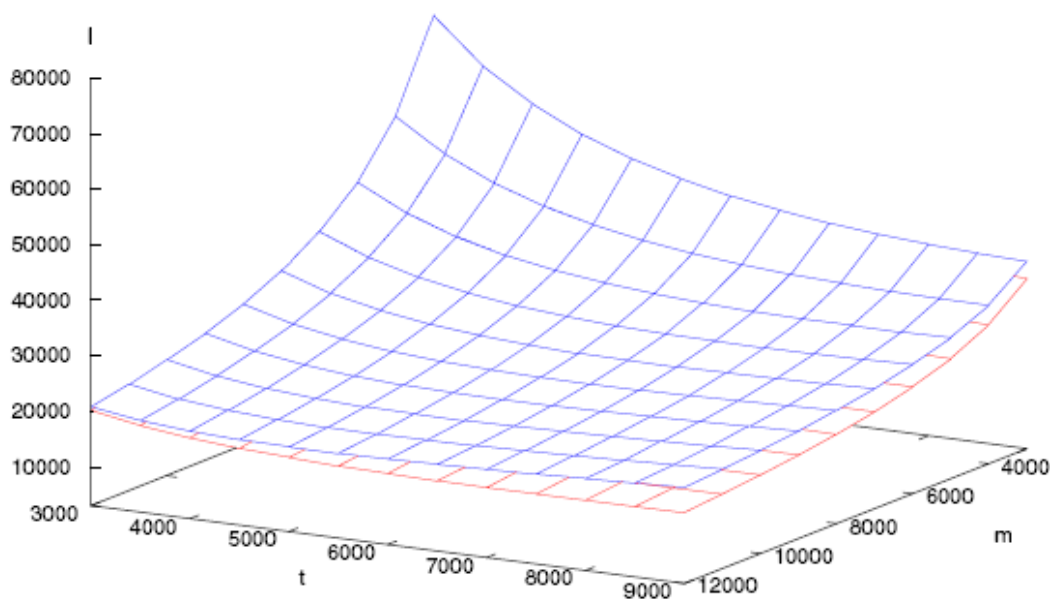


Рисунок 3. Сравнение вероятности успеха классических таблиц и Rainbow-таблиц. Верхняя плоскость представляет ограничение в 99.9% успеха для классических таблиц, нижняя – то же ограничение, но для Rainbow-таблиц. Шкала для Rainbow-таблиц была настроена таким образом, что можно провести прямое соотношение обоих типов таблиц: $m \rightarrow m'/t, I \rightarrow I'/t$

Основываясь на рисунке 1 мы установили параметры для классических таблиц:

$$t_c = 4666, m_c = 8192;$$

для Rainbow-таблиц:

$$t_r = 4666, m_r = t_c \times m_c = 38'223'872.$$

Мы сгенерировали 4666 классических таблиц и одну Rainbow-таблицу и замерили вероятность успеха, взломав 500 случайных паролей на стандартном компьютере (P4 1.5GHz, 500Mb RAM). Результаты приведены в таблице ниже:

	Классическая таблица с ВТ	Rainbow-таблица
t, m, I	4666, 8192, 46664666	4666, 38'223'872, 1
Прогнозируемый охват	75.5%	77.5%
Замеренный охват	75.8%	78.8%

Таблица 1. Замеренный охват для классических таблиц с выделенными точками и Rainbow-таблиц по результатам взлома 500 хэшей к паролям.

Этот эксперимент наглядно демонстрирует, что Rainbow-таблицы могут достичь той же вероятности успеха с теми же затратами памяти, что и классические таблицы. Зная это, любопытно сравнить время криптоанализа обоих методов, обнаружив, что с использованием Rainbow-таблиц поиск паролей идет в два раза быстрее. В таблице 2 мы сопоставляем среднее время на криптоанализ со средним количеством операций с хэшами и ложными тревогами, возникшими за время криптоанализа:

	Классическая таблица с ВТ	Rainbow-таблица	Коэффициент
<i>t, m, l</i>	4666, 8192, 4666	4666, 38'223'872, 1	1
	Среднее время на криптоанализ		
При успехе	68.9 сек.	9.37 сек.	7.4
При неудаче	181.0 сек.	26.0 сек.	7.0
В среднем	96.1 сек.	12.9 сек.	7.4
	Среднее количество вычисленных хэшей		
При успехе	48.3 млн.	6.77 млн.	7.1
При неудаче	126 млн.	18.9 млн.	6.7
В среднем	67.2 млн.	9.34 млн.	7.2
	Среднее количество попыток поиска		
При успехе	1779	2136	0.83
При неудаче	4666	4666	1
В среднем	2477	2673	0.93
	Среднее кол-во найденных подходящих конечных точек		
При успехе	1034	620	1.7
При неудаче	2713	2020	1.3
В среднем	1440	917	1.6
	Среднее количество ложных тревог		
При успехе	4157	1492	2.8
При неудаче	10913	5166	2.1
В среднем	5792	2271	2.6
	Среднее кол-во вычисленных хэшей на ложную тревогу		
При успехе	9622	3030	3.2
При неудаче	9557	1551	6.2
В среднем	9607	2540	3.8

Таблица 2. Сводная статистика классических таблиц с выделенными точками и Rainbow-таблиц.

Из таблицы 2 видно, что наш метод в действительности примерно в 7 раз быстрее оригинального. На самом деле, за каждый сеанс криптоанализа производится примерно 9.3 млн. вычислений хэшей с использованием улучшенного метода, в то время как оригинальный метод производит 67.2 млн. вычислений. Множи-

тель на два вытекает из структуры таблиц (см. выше). Дополнительный прирост скорости вызван тем фактом, что с использованием выделенных точек возникает значительно больше т.н. ложных тревог (в среднем в 2.8 раза больше), следовательно, увеличивается временные затраты на их обработку. Оба эффекта вызваны тем обстоятельством, что длина цепочек при использовании выделенных точек непостоянна.

4.1. Важность постоянства

Неизбежное притяжение: Вариации длины цепочек ведут к вариациям вероятности слияния. В пределах данного набора цепочек (например, одной таблицы) более длинные цепочки будут иметь больше шансов слиться с другими, нежели короткие. Соответственно, слияния будут создавать большее дерево из более длинных цепочек и меньшее из более коротких. Негативный эффект удваивается в случае ложной тревоги. Вероятность ложной тревоги больше в больших деревьях, поскольку в большом дереве больше вероятность слияния, чем в малом. Одно слияние в большом дереве создает больше ложных тревог, поскольку в нем больше цепочек, и для проверки ложной тревоги необходимо регенерировать все цепочки. Таким образом, ложные тревоги имеют тенденцию возникать не только в длинных цепочках, но и в их больших наборах.

Большие затраты: Кроме действия эффекта притяжения в более длинных цепочках, количество вычислений, необходимых для подтверждения ложной тревоги, увеличивается при переменной длине цепочек по сравнению с цепочками постоянной длины. Когда длина цепочки неизвестна, то для подтверждения ложной тревоги необходимо воссоздать всю цепочку. Работая с цепочками постоянной длины, мы можем подсчитать количество вычислений, произведенных до конца цепочки и уже тогда точно знать – в какой позиции ожидать ключ. Таким образом, для подтверждения ложной тревоги нам нужно сгенерировать только лишь сектор цепочки. Более того, с Rainbow-цепочками ложные тревоги встречаются чаще во время поиска в более длинных цепочках (т.е. в столбцах в левой части таблицы). К счастью, именно здесь для подтверждения ложной тревоги и требуется восстановить кратчайший отрезок цепочки.

Оба эти эффекта наглядно представлены в таблице 2 количеством найденных конечных точек, количеством ложных тревог и количеством вычислений на каждую ложную тревогу в случае неудачи. Работая с ВТ каждая подходящая точка создает примерно 4 ложных тревоги и средняя длина генерируемых цепочек составляет примерно 9600. С Rainbow-цепочками на каждую найденную конечную точку приходится лишь 2.5 ложных тревоги и на каждую ложную тревогу генерируется всего 1500 ключей.

Тот факт, что более длинные цепочки ведут к образованию большего числа слияний был отмечен в [7], но без указания на то, что это увеличивает вероятность ложных тревог и затрат на них. В итоге авторы предлагаю лишь использовать цепочки в пределах некоторого диапазона длин. Это сокращает вышеописанные проблемы, связанные с различной длиной цепочек, однако сокращается и охват, достигаемый одной функцией свертки, увеличивая при этом затраты на предварительный расчет.

4.2 Дальнейшее увеличение преимущества

Мы рассчитали прогнозируемое преимущество Rainbow-таблиц перед классическими таблицами, рассмотрев худший случай, когда ключ ищется во всех столбцах Rainbow-таблиц и без учета ложных тревог. В то время как количество вычислений в ходе поиска по Rainbow-таблице увеличивается квадратично от 1 к $(t^2 - 1) / 2$, а при поиске в классических таблицах оно линейно увеличивается до t^2 . Поэтому, если ключ найден на ранних стадиях, то преимущество может быть еще существеннее (оно зачастую зависит от множителя t). Этот дополнительный выигрыш практически сводится на "нет" тем обстоятельством, что в Rainbow-таблицах ложные тревоги, случающиеся в начале поиска (которые достаточно редки), при этом являются наиболее ресурсоемкими. И все же должна быть воз-

возможность сконструировать случай (почти невероятный), когда Rainbow-таблицы дают заведомо большое преимущество перед классическими. Единственный способ это сделать – требование вероятности успеха, максимально приближенного к 100%, и большое значение t . Примеры в литературе нередко исходят из вероятности успеха до 80% с $N^{1/3}$ таблиц на $N^{1/3}$ цепочек с $N^{1/3}$ точек. Такая конфигурация может быть заменена одной Rainbow-таблицей на $N^{2/3}$ рядов с $N^{1/3}$ ключей. Для некоторых задач вероятность успеха в 80% может быть достаточной, особенно если есть несколько примеров зашифрованного текста и требуется восстановить просто любой ключ.

В нашем примере (восстановление LM-пароля) мы, как правило, заинтересованы в одном конкретном пароле (например, администраторском). В этом случае нам нужна практически абсолютная уверенность в успехе. Высокие шансы успеха требуют конфигурации, в которой количество таблиц в несколько раз превышает длину цепочек. Таким образом, мы приходим к нескольким Rainbow-таблицам (в нашем случае – 5). Требование высокой вероятности успеха приводит к тому, что ключ зачастую находится на ранних стадиях и редко требуется производить поиск по всем рядам всех таблиц. Чтобы воспользоваться этим обстоятельством, мы должны удостовериться, что ищем в пяти Rainbow-таблицах не последовательно в каждой таблице, а сначала в последних столбцах всех таблиц и только затем переходим к предпоследним. В результате этой процедуры мы приобретаем 12-кратное преимущество, используя пять таблиц для достижения успеха с вероятностью 99.9%, по сравнению с 7-кратным преимуществом при использовании одной таблицы и вероятностью успеха 78%. В следующем разделе мы рассмотрим это подробнее.

4.3 Взлом Windows-паролей за секунды

Заметив, что Rainbow-цепочки выигрывают в производительности у классических, мы создали большой комплекс таблиц для достижения нашей цели – вероятности успеха в 99.9%. Из измерений первой таблицы вытекает, что нам понадобится 4.45 таблиц на 38223872 строки и 4666 столбцов. Чтобы иметь целое число таблиц и соответствовать ограничениям памяти в 1.4GB мы решили сгенерировать 5 таблиц на 35'000'000 строк. С другой стороны мы сгенерировали 23'330 таблиц на 4666 столбцов и 7501 строку. Результаты даны в таблице 3. Мы взломали 500 паролей со 100% успехом в обоих случаях.

	Классическая таблица с VT	Rainbow-таблица	Кэфф.	Последовательность Rainbow	Кэфф.
t, m, l	4666, 7501, 23330	4666, 35млн., 5	1	4666, 35млн., 5	1
время криптоанализа	101.4с	66.3	1.5	13.6с	7.5
вычислений хэшей	90.3млн.	7.4млн.	12	11.8млн.	7.6
ложных тревог (лт)	7598	1311	5.8	2773	2.7
Хэшей на лт	9568	4321	2.2	3080	3.1
затраты на лт	80%	76%	1.1	72%	1.1
вероятность успеха	100%	100%	1	100%	1

Таблица 3. Статистика криптоанализа с набором таблиц для достижения вероятности успеха в 99.9%. Средний столбец наглядно показывает, что при использовании Rainbow-таблиц требуется в 12 раз меньше вычислений. Временные затраты на криптоанализ дают выигрыш лишь в 1.5 раза из-за обращений к дисковой памяти. На машине с 500Mb RAM больший выигрыш во времени (в 7.5 раз) может быть достигнут ограничением поиска одновременно лишь по одной Rainbow-таблице.

Из таблицы 3 видно, что Rainbow-таблицы в 12 раз сокращают количество вычислений по сравнению с классическими таблицами с выделенными точками. К сожалению, выигрыш во времени составляет лишь в 1.5 раза. Это происходит из-за произвольного обращения к 1.4GB данных на компьютере с 500MB RAM. В предыдущих измерениях с единой таблицей таблица могла оставаться в кэше файловой системы, что неосуществимо при работе с памятью. Вместо расширения памяти рабочей станции до 1.5GB RAM мы предпочли реализовать подход с

последовательным поиском в каждой таблице. Это позволило нам проиллюстрировать выводы в конце предыдущего раздела. При одновременном поиске ключа по всем таблицам вместо последовательного перебора всех цепочек мы работаем с более короткими цепочками и, следовательно, выполняем меньше работы (7.4 млн. операций вместо 11.8 млн.). Более короткие цепочки также означают и меньшее количество ложных тревог (1311 на каждый взломанный ключ вместо 2773). Но короткие цепочки помимо этого означают и большее число вычислений для подтверждения ложной тревоги (4321 вместо 3080). Любопытно, что во всех случаях 75% всех затрат на криптоанализ приходится на вычисления, связанные с ложными тревогами.

Рассматривая общие параметры метода, мы также отмечаем, что предварительные вычисления таблиц в 10 раз превышают затраты на составление полного словаря. Столь значительные затраты являются результатом вероятностной сущности метода и могут быть сокращены в три раза по сравнению с полным словарем, если принять вероятность успеха, равную 90% вместо 99.9%.

5 Перспективы идеальной таблицы

И Rainbow-таблицы и классические таблицы с ВТ обладают свойством обнаружения сливающихся цепочек по идентичным конечным точкам. Поскольку таблицы в любом случае сортируются по конечным точкам, представляется перспективным создание идеальных таблиц, удаляя все цепочки, которые сливаются с уже существующими в таблице. В случае с выделенными точками мы даже можем выбрать сохранение самой длинной из сливающихся цепочек с тем, чтобы увеличить охват таблицы. Вероятность успеха работы с Rainbow-таблицами и таблицами с ВТ легко подсчитать, по крайней мере, с допущением средней длины цепочек с выделенными точками. В этом случае очевиден равный шанс успеха Rainbow-таблицы размера $mt \times t$ в сравнении с t таблиц размера $m \times t$. Действительно, в предыдущем случае у нас было t строк по mt отдельных ключей, в то время как в последнем – t таблиц по mt ключей каждая.

В идеале нам бы хотелось построить единую идеальную таблицу, охватывающую весь домен N ключей. Задача идеальной таблицы – предсказать, сколько не сливающихся цепочек длины t можно сгенерировать. Для Rainbow-цепочек этот показатель может быть подсчитан тем же путем, каким мы вычисляем вероятность успеха для неидеальных таблиц. После оценки количества выделенных точек в каждом столбце таблицы нам останется только посмотреть на число выделенных точек в последнем столбце, чтобы узнать, сколько будет отдельных цепочек.

$$\hat{P}_{table} = 1 - e^{-t \frac{m_1}{N}} \quad (4)$$

где

$$m_1 = N$$

и

$$m_{n+1} = N \left(1 - e^{-\frac{m_n}{N}} \right)$$

Для цепочек, разделенных выделенными точками, эти вычисления несколько более сложны. Из-за неизбежного притяжения, описанного выше, более длинные цепочки будут слиты в большие деревья. Следовательно, устраняя сливающиеся цепочки, мы скорее будем удалять более длинные, чем короткие. Единственный эксперимент с 16 миллионами цепочек длиной в 4666 показал, что после удаления всех слияний (с сохранением самых длинных цепочек) осталось только 2% цепочек, и средняя длина их сократилась с 4666 до 386! Для сохранения средней длины в 4666 нам потребуется удалить 96% оставшихся цепочек, чтобы сохранить 4% (14060) самых длинных.

Затраты на предварительный расчет при генерации идеальных таблиц максимального размера непозволительно (Nt). И наша задача выполняется лишь с использованием комплекса меньших таблиц.

Мы как раз сейчас занимаемся более подробным разбором идеальных таблиц. Предполагаем, что в связи с ограниченностью возможного числа несливающихся цепочек целесообразным будет использовать близкие к идеальным таблицы.

6. Заключение

Мы представили новый способ генерации предварительных данных для оригинального метода Hellman'a – компромисса между временем и памятью в ходе криптоанализа. Наша оптимизация обладает тем же свойством, что и использование ВТ, т.е. сокращение числа обращений к таблице на индекс длины цепочек. При равной вероятности успеха наш метод вдвое сокращает количество вычислений, необходимых для криптоанализа по сравнению с оригинальным методом и на еще более существенный показатель (в нашем эксперименте 12) по сравнению с методом ВТ. Мы доказали, что причиной этого дополнительного выигрыша является переменная длина цепочек, разделенных выделенными точками, что приводит к увеличению доли ложных тревог и дополнительных затрат на каждую ложную тревогу. Предполагаем, что при других параметрах (например, более высокой вероятности успеха) выигрыш может оказаться и больше 12, нежели мы получили в нашем эксперименте. Все это делает наш метод довольно привлекательной заменой оригинального метода, улучшая использование выделенных точек.

Тот факт, что наш метод ведет к получению цепочек постоянной длины, также значительно упрощает анализ метода по сравнению с цепочками переменной длины с использованием выделенных точек. Он также позволяет избежать лишних затрат на предварительный расчет, возникающих из-за необходимости отбрасывать цепочки переменной длины из-за несоответствия длины или содержащегося в них цикла (петли). Постоянная длина также доказывает свою выигрышность в плане производительности аппаратных средств.

Наконец, наш эксперимент наглядно продемонстрировал, что компромисс времени и памяти позволяет любому владельцу современного персонального компьютера взломать криптосистемы, считавшиеся безопасными на момент их создания многие годы назад и до сих пор использующиеся. Это наглядное свидетельство важности постепенной ликвидации старых криптографических систем и заменой их на лучшие, при их наличии. В частности, поскольку память для этого типа взлома так же важна, как и скорость обработки, то обычные компьютеры вдвойне выигрывают от технологического прогресса.

Ссылки

1. J. Borst, B. Preneel, and J. Vandewalle. On time-memory tradeoff between exhaustive key search and table precomputation. In P. H. N. de With and M. van der Schaar-Mitrea, editors, *19th Symp. on Information Theory in the Benelux*, pages 111-118, Veldhoven (NL), 28-29 1998. Werkgemeenschap Informatie- en Communicatietheorie, Enschede (NL).
2. D.E. Denning. *Cryptography and Data Security*, page 100. Addison-Wesley, 1982.
3. Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *STOC 1991*, pages 534-541, 1991.
4. M. E. Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, IT-26:401-406, 1980.
5. Kim and Matsumoto. Achieving higher success probability in time-memory tradeoff cryptanalysis without increasing memory size. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 1999.
6. Koji KUSUDA and Tsutomu MATSUMOTO. Optimization of time-memory tradeoff cryptanalysis and its application to DES, FEAL-32, and skipjack. *IEICE Transactions on Fundamentals*, E79-A(1):35-48, January 1996.
7. F.X. Standaert, G. Rouvroy, J.J. Quisquater, and J.D. Legat. A time-memory tradeoff using distinguished points: New analysis & FPGA results. In *proceedings of CHES 2002*, pages 596-611. Springer Verlag, 2002.

7 Приложение

Вероятность успеха с использованием одной Rainbow-таблицы может быть рассчитана, если рассматривать каждый столбец таблицы с точки зрения проблемы классического использования. Начинаем с $m_1 = m$ отдельных ключей в первом столбце. Во втором столбце ключи m_1 произвольно распределены по ключевому пространству размера N , генерируя m_2 отдельных ключей:

$$m_2 = N \left(1 - \left(1 - \frac{1}{N} \right)^{m_1} \right) \approx N \left(1 - e^{-\frac{m_1}{N}} \right)$$

В каждом столбце I содержится m_i отдельных ключей. Вероятность успеха таблицы, таким образом:

$$P = 1 - \prod_{i=1}^t \left(1 - \frac{m_i}{N} \right)$$

где

$$m_1 = m$$

и

$$m_{n+1} = N \left(1 - e^{-\frac{m_n}{N}} \right)$$

Это результат – не конечный и может быть рассчитан в числах. Вероятность успеха не уступает классическим таблицам, поскольку большое число неразложимых составляющих в сумме этого уравнения требует цифровой интерполяции.

Тот же подход может быть использован для расчета возможного числа не сливающихся сгенерированных цепочек. Поскольку сливающиеся цепочки распознаются по идентичным конечным точкам, число отдельных ключей в последнем столбце m_t является числом несливающихся цепочек.

Максимальное число цепочек достигается назначением каждого отдельного ключа в ключевом пространстве N в качестве стартовой точки.

$$m_1 = N, m_{n+1} = N \left(1 - e^{-\frac{m_n}{N}}\right)$$

Вероятность успеха в таблице с максимальным числом несливающихся цепочек составляет:

$$\hat{P} = 1 - \left(1 - \frac{m_t}{N}\right)^t \approx 1 - e^{-t \frac{m_t}{N}}$$

Отметьте, что затраты на построение такой таблицы составляют Nt .

Прим. переводчика: на самом деле, эта функция подсчитывает вероятность несколько неверно и ниже дан исправленный вариант подсчета вероятности успеха в формате языка MatLAB:

```
function ret = calc_success_probability(N, t, m)
arr = zeros(1, t - 1);
arr(1) = N * (1 - exp(-m/N));
for i = 2 : t - 1
    arr(i) = N * (1 - exp(-arr(i-1)/N));
end;
ret = 1;
for i = 1 : t - 1
    ret = ret * (1 - arr(i)/N);
end;
ret = 1 - ret;
```

Java-скрипт <http://www.OutsidePro.com/rainbow.php> для расчета параметров Rainbow-таблиц использует уже исправленную версию этой формулы.